WTclient DLL User's Guide

The WinTECH Software Rapid Client Development DLL, (WTclient), provides an easy to use API for integrating a custom application with data from any OPC Server. All the details of COM and OPC are handled by the DLL, which allows an application to easily obtain data points from a Server, without having to be concerned with the actual implementation of the underlying interfaces. The DLL may be easily integrated with existing applications, or new ones. All required OPC Interfaces are supported for both OPC 1.0 and OPC 2.0 Data Access Standards as well as the Browse Interface and support for connecting to a simple Alarms & Events Server.

This user's guide should be used as a companion document to the ***OPC Data Access*** and ***OPC Alarms & Events*** Specifications available from the OPC Foundation web-site at http://www.opcfoundation.org.

Creating a Custom OPC Client using WTclient.DLL

Link WTclient.lib with the Application

WTclient.lib contains the export definitions for the DLL's API. Include this file with the project files for the custom application and include WTclientAPI.h with those modules which will be making calls into the DLL. If programming in Visual Basic, include the module1.bas from the VBA_example, (supplied with the demo), into your project.

DCOM Initialization

Beginning with WTClient.DLL Version 2.0, it is now the Client Application's responsibility to properly initialize DCOM. Prior to Version 2.0, WTClient.dll performed DCOM and Security initialization in the DLLMain function and called CoUninitialize when the dll was unloaded. This presented some problems due to the load-order of various Windows dll's. These initialization calls have been moved to a new exported function, (WTclientCoInit ()), that the user application can call during the startup of the main executable thread. WTclientCoInit () initializes DCOM as multi-threaded and uses default security. Your application may choose to perform DCOM initialization itself rather than using the exported function. In either case, your application MUST call CoUninitialize() when terminating.

Obtaining the List of OPC Servers

WTclient.dll exports two functions that allow the controlling application to obtain a list of available OPC Servers.

int NumberOfOPCServers (BOOL UseOPCENUM, LPCSTR MachineName);
BOOL GetServerName (int index, LPSTR Buffer, int BufSize);

OPCENUM is a component supplied by the OPC Foundation to allow prospective client applications to obtain a list of available servers on the local or remote machine. Unfortunately, at the time of this writing, the operation of OPCENUM is not consistant across all platforms. WTclient.dll allows the application to attempt to use the OPCENUM component to obtain the list of Servers, or to bypass OPCENUM and obtain the list by scanning the local copy of the Windows Registry. If OPCENUM is used, you can pass the name of a remote machine to obtain the remote server list.

Once the number of servers has been returned from WTclient.dll, the application can iterate through the list of names by calling:

BOOL GetServerName (int index, LPSTR Buffer, int BufSize);

A user supplied character buffer receives a name from the Server list as identified by a passed index. BufSize identifies the length of the user supplied character buffer and prevents the dll from loading a name that's too long. GetServerName returns TRUE if a valid Server name is returned in Buffer, otherwise the return value is FALSE.

A complementary set of functions allow the application to obtain a list of available Alarms & Events Servers.

int NumberOfOPC_AEServers (LPCSTR MachineName);
BOOL GetOPC_AEServerName (int index, char *pBuf, int BufSize);

Establishing an OPC Connection

HANDLE ConnectOPC(LPCSTR MachineName, LPCSTR ServerName, BOOL EnableDLLBuffering);

HANDLE ConnectOPC1(LPCSTR MachineName, LPCSTR ServerName, BOOL EnableDLLBuffering);

HANDLE ConnectOPC_AE(LPCSTR MachineName, LPCSTR ServerName);

This function returns a valid HANDLE if a connection could be established to the OPC Server defined by MachineName and ServerName. Multiple simultaneous connections to different servers may be established, and the returned HANDLE identifies the connection for future calls to create groups and items. Passing a Null String as the MachineName parameter identifies the local machine. WTclient will attempt the connection using the OPC 2.0 ConnectionPoint Interface. If not available, it will revert the the OPC 1.0 DataObject Interface.

WTclient.dll operates in basically two modes. The first is used with those applications which support 'C-Style' callback functions. In this case, no buffering is performed inside the dll. As data change notifications are received from the connected OPC Server, callbacks are used to pass the new information to the controlling application, (refer to EnableOPCNotification below)

For those applications designed using tools which do not support callback functions, (such as Visual Basic 5.0), WTclient may be configured to maintain a list of all OPC Items, (Tags), created by the application. As data from the Server changes, the corresponding tag value in the dll's list will be updated. The controlling application may read the value of a tag at any time, (refer to ReadOPCItem below).

ConnectOPC1 will always attempt an OPC 1.0 connection.

ConnectOPC_AE will establish a connection to the OPC Alarms & Events Server specified by MachineName and ServerName.

void DisconnectOPC(HANDLE hConnect);
void DisconnectOPC_AE(HANDLE hConnect);

When an application terminates, it is responsible for disconnecting from an attached Server. The DisconnectOPC(), (or Disconnect_OPCAE()), function will provide a clean shutdown of the connection defined by hConnect.

Obtaining a list of Server Tag Names

int NumberOfOPCItems(HANDLE hConnect);
BOOL GetOPCItemName (HANDLE hConnect, int index, char *pBuf, int BufSize);

These two functions operate similarly to those which allow you to obtain the list of Server names. NumberOfOPCItems() returns the total number of unique tags supported by a connected OPC Server. Incrementally calling GetOPCItemName will step through all the available tag names to allow the controlling application to select an item or present a Browse list to the user. NumberOfOPCItems will browse the complete list of item names from the server and present the list in a 'FLAT' format.

BOOL GetNameSpace (HANDLE hConnect, WORD *pNameSpace);

BOOL BrowseTo (HANDLE hConnect, LPCSTR NodeName);

char SetWTclientQualifier (char qualifier);

int BrowseItems (HANDLE hConnect, WORD Filter);

To provide better access to a server with a hierarchical name space, the WTClient dll provides functions to allow the application to browse item names directly from the server. GetNameSpace allows the application to determine the namespace of the server. BrowseTo moves the current browse position to the name specified, and BrowseItems fills the internal name array with node names from the server. BrowseTo may be used with either OPC_LEAF to read the leaf names of OPC_BRANCH to read the branch names immediately below the current browse position. The names may then be retrieved using the GetOPCItemName function. The WtClient DLL maintains a single name array for use by NumberOfOPCItems and BrowseItems so the application must be careful when browsing the server namespace, not to overwrite values from a previous call. During iteraction of the hierarchical tree, the all item names must be copied into local storage immediately after calling BrowseItems.

Calling BrowseItems from the Root position using OPC_FLAT behaves identically to NumberOfOPCItems().

SetWTclientQualifier allows the application to change the delimiting character used to parse the nodes of a hierarchical namespace. The default delimiter expected by the client dll is '.', however, some servers use the '.' character as part of the node name.

BOOL SetBrowseFilters (HANDLE hConnect, LPCSTR UserString, VARTYPE DataType, DWORD AccessType);

Browse filters may be set by the application to effect the number of OPC Items returned from the server. The application may choose to retrieve only those items of a certain data type or access rights.

Creating OPC Groups and Tags

HANDLE AddOPCGroup (HANDLE hConnect, LPCSTR Name, DWORD *pRate, float *pDeadBand);
void RemoveOPCGroup (HANDLE hConnect, HANDLE hGroup);

WTclient.dll exports two functions which allow you to create and remove an OPC Group. The AddOPCGroup() function returns a handle to uniquely identify the group, and must be supplied to other Wtclient functions to reference tags, etc. The values passed to AddOPCGroup as parameters are determined by the controlling application and are specific to each Group. The name is arbitrary and not used by the Server. pRate is a pointer to the desired refresh rate, (in milliseconds), that the server uses to provide updates back to the client. The actual rate is returned from the server in pRate.

BOOL ChangeOPCGroupState (HANDLE hConnect, HANDLE hGroup, BOOL Active));

The defined OPC Group may be activated, (Active = TRUE), or deactivated, (Active = FALSE), using this function.

**HANDLE AddOPCItem (HANDLE hConnect, HANDLE hGroup, LPCSTR ItemName);
**
void RemoveOPCItem (HANDLE hConnect, HANDLE hGroup, HANDLE hItem);

To add an item to a defined OPC Group, simply call the AddOPCItem function with the group's handle and the tag name of the desired item. A handle value will be returned to uniquely define the item, INVALID_HANDLE_VALUE will be returned if the requested item does not exist). Call RemoveOPCItem to clean-up allocated memory when closing a group.

BOOL SetItemUpdateHandle (HANDLE hConnect, HANDLE hGroup, HANDLE hItem, HANDLE hUpdate);

SetUpdateHandle allows the client application to specify a unique handle to be used by the server during Data Update Callbacks. This function is provided as a convenience to the client application for more efficient design or handing new data presented by the server.

BOOL GetOPCItemNameFromHandle (HANDLE hConnect, HANDLE hGroup, HANDLE hItem, char *pBuf, int BufSize);

This function allows the client application to retrieve the OPC Item name by passing the handle returned from AddOPCItem.

BOOL GetOPCItemType (HANDLE hConnect, HANDLE hGroup, LPCSTR ItemName, VARTYPE *pType, DWORD *pAccessRights);

This function returns the native variant type and access rights for the specified item.

int NumberOfItemProperties (HANDLE hConnect, LPCSTR ItemName);

This function returns the number of OPC Item properties associated with the specified tag.

BOOL GetItemPropertyDescription (HANDLE hConnect, int PropertyIndex, DWORD *pPropertyID, VARTYPE *pVT, BYTE *pDescr, int BufSize);

A description of each item property may be obtained by iterating through the list created by NumberOfOPCItemProperties.

BOOL ReadPropertyValue (HANDLE hConnect, LPCSTR Itemname, DWORD PropertyID, VARIANT *pValue);

The current value of an Item Property is returned by ReadItemProperty.

Getting updated Tag Values from the Server

BOOL EnableOPCNotification (HANDLE hConnect, NOTIFYPROC lpCallback);

If your application supports callbacks, WTclient will issue the callback when notification from a connected Server indicates a change in the value of a Tag's data. The prototype for the callback is defined as follows:

void CALLBACK EXPORT OPCUpdateCallback (HANDLE hGroup, HANDLE hItem, VARIANT *pVar, FILETIME timestamp, DWORD quality)

The current value for a tag, timestamp and OPC Quality flags will be supplied to the application via the callback. The Tag is identified by the associated handles identifying the group and item.

BOOL ReadOPCItem (HANDLE hConnect, HANDLE hGroup, HANDLE hItem, VARIANT *pVar, FILETIME *pTimeStamp, DWORD *pQuality);

If WTclient has been configured to maintain an internal buffer of tags, the controlling application may read the current value of a tag by using the ReadOPCItem function. If the requested item is not available, (or if DLLBuffering has not been selected), the function will return FALSE.

HRESULT ReadOPCItemFromDevice (HANDLE hConnect, HANDLE hGroup, HANDLE hItem, VARIANT *pVar, FILETIME *pTimeStamp, DWORD *pQuality);

This function performs a synchronous read directly from the server for the specified item.

Writing Tag Values to the Server

DWORD WriteOPCItem (HANDLE hConnect, HANDLE hGroup, HANDLE hItem, VARIANT *pVar, BOOL DoAsync);

Writing new data to the connected Server is accomplished via the WriteOPCItem function. The identifying handles for the group and item must be supplied as well as the data to be written. The DoAsync parameter instructs the dll to perform an Asynchronous write to the Server. For ***Synchronous*** writes, the return value indicates the hResult returned from the server and a ***non-zero value indicates an error***. For ***Asynchronous*** writes, the return value is a Transaction ID that defines the asynchronous callback handle and ***a zero value indicates an error.***

Miscellaneous Functions and callbacks

**BOOL GetSvrStatus (HANDLE hConnect, OPCSERVERSTATUS *pSvrStatus,
int VendorInforBufSize);**

The current status of the identified server is returned in the supplied buffer.

BOOL SetClientName (HANDLE hConnect, LPCSTR Name);

This function allows the client to define a name to designate the connection with the server.

BOOL EnableErrorNotification (ERRORPROCAPI lpCallback);

The WTClient dll can notify the client application of errors that occur during processing of data interface calls to an OPC Server. Error messages may be generated by the dll if unexpected situations occur during processing of a user request. The default operation of the dll is to display a user message in a dialog box describing the condition. In most cases, it is more appropriate for the application itself to handle the error message rather than having the dialog generated by the dll. If Error Notification is enabled, control will be passed to the application via the following callback:

void CALLBACK EXPORT ErrorMsgCallback (DWORD hResult, char *pMsg)

A textual description of the error is contained in pMsg along with the resultant hResult.

BOOL EnableClientEventMsgs (EVENTMSGPROC lpCallback);

The WTClient dll can also notify the client application of various events as they occur during operation of the connection with the server. These events are basically used for debugging and represent normal activity associated with processing individual interfaces, (such as entering and exiting a particular function). Generally, the client application does not need to be made aware of theses events, however, if it wishes to provide a low level description of how the connection is functioning, it can receive these debugging statements as they occur by enabling Event Msgs.

void CALLBACK EXPORT EventMsgCallback (char *pMsg)

A textual description of the event is contained in pMsg.

BOOL EnableShutdownNotification (HANDLE hConnect, SHUTDOWNPROCAPI lpCallback);

If a connected OPC Server wishes to shutdown, it may request that any or all clients disconnect. The Client application should enable the shutdown notification within the WTClient dll to handle this request and terminate the connection.

void CALLBACK EXPORT ShutdownCallback (HANDLE hConnect)

The hConnect parameter identifies the server connection that is requesting the disconnect..

Creating an Alarms & Events Subscription

BOOL Create_AE_Subscription (HANDLE hConnect, HANDLE SubscriptionHandle,
DWORD *pBufferTime, DWORD *pMaxSize);

BOOL WINAPI EnableAECallback (AE_PROC lpCallback);
BOOL WINAPI EnableExtendedAECallback (AE_PROC_EX lpCallback);

To begin receiving alarm and event messages from the connected server, a subscription must be in initiated and a callback procedure defined. The application passes a SubscriptionHandle that the server uses to identify the connection to the callback. pBufferTime and pMaxSize refer to the requested update frequency and the maximum number of events to be exchanged at one time. *(Note: These values are for efficiency purposes only. If the standard callback is used, the WTClient DLL will set up the connection and callback to the server. The DLL will always issue a single callback to the application per event message, regardless of how many event messages are in the server callback to the dll. If the extended version of the callback is used, these parameters have the effect as described in the OPC A&E Specification with respect to the ONEVENTSTRUCT.)*

One of two callback procedures may be defined, depending upon the complexity of the client application. Teh standard callback procedure is prototyped as follows:

void CALLBACK EXPORT AECallback (HANDLE SubscriptionHandle, char *pSource,
FILETIME timestamp, char *pMsg, DWORD severity)

The SubscriptionHandle is the same as created by the subscription, pSource identifies the source of the event and pMsg is the description.

If additional information regarding the alarm & event notification, the client must enable the extended version of the callback:

void CALLBACK EXPORT ExtendedAECallback (HANDLE SubscriptionHandle, BOOL bRefresh,
BOOL bLastRefresh, DWORD dwCount, ONEVENTSTRUCT *pEvents)

Notice that the extended version of the A&E Callback contains the unfiltered parameter list as returned by the server OnEvent Interface, and may contain more than one event message.

A&E Server Direct Interface Calls

The following functions are provided to allow the client application to call directly into the attached A&E Server. The parameter list is passed directly to the associated Server Interface without change and the resulting returned values originate from the server itself. The WTClient.dll does not modify these parameters in any way and their definition is described in the OPC Alarms & Events Specification available from the OPC Foundation. Please refer to this specification for a detailed description of each of the following functions.

BOOL GetAESvrStatus (HANDLE hConnect, OPCEVENTSERVERSTATUS *pSvrStatus,
int VendorInforBufSize);

The current status of the identified server is returned in the supplied buffer.

BOOL Refresh_AE_Subscription (HANDLE hConnect, HANDLE hSubscription);

This function allows the client to request a refresh of the defined A&E subscription from the server.

```
HRESULT  AckCondition (HANDLE      hConnect,
                       DWORD       dwCount,
                       LPWSTR      szAcknowledgerID,
                       LPWSTR      szComment,
                       LPWSTR      *pszSource,
                       LPWSTR      *pszConditionName,
                       FILETIME    *pftActiveTime,
                       DWORD       *pdwCookie,
                       HRESULT     **ppErrors);
```

The client uses the AckCondition method to acknowledge one or more conditions in the Event Server. The client receives event notifications from conditions via the IOPCEventSink::OnEvent callback. This AckCondition method specifically acknowledges the condition becoming active or transitioning into a different sub-condition (and no other state transition of the condition). One or more conditions belong to a specific event source – the source of the event notification. For each condition-related event notification, the corresponding Source, Condition Name, Active Time and Cookie is received by the client as part of the OnEvent callback parameters. ***(Refer to the OPC Alarms & Events Specification for a detailed description of the parametr list and return value.)***

HRESULT EnableConditionByArea (HANDLE hConnect,
DWORD dwNumAreas,
LPWSTR *pszAreas);

Places all conditions for all sources within the specified process areas into the enabled state. Therefore, the server will now generate condition-related events for these conditions.

The effect of this method is global within the scope of the event server. Therefore, if the server is supporting multiple clients, the conditions are enabled for all clients, and they will begin receiving the associated condition-related events. ***(Refer to the OPC Alarms & Events Specification for a detailed description of the parametr list and return value.)***

HRESULT EnableConditionBySource (HANDLE hConnect,
DWORD dwNumSources,
LPWSTR *pszSources);

Places all conditions for the specified event sources into the enabled state. Therefore, the server will now generate condition-related events for these conditions.

The effect of this method is global within the scope of the event server. Therefore, if the server is supporting multiple clients, the conditions are enabled for all clients, and they will begin receiving the associated condition-related events. ***(Refer to the OPC Alarms & Events Specification for a detailed description of the parametr list and return value.)***

HRESULT DisableConditionByArea (HANDLE hConnect,
DWORD dwNumAreas,
LPWSTR *pszAreas);

Places all conditions for all sources within the specified process areas into the disabled state. Therefore, the server will now cease generating condition-related events for these conditions.

The effect of this method is global within the scope of the event server. Therefore, if the server is supporting multiple clients, the conditions are disabled for all clients, and they will stop receiving the associated condition-related events. ***(Refer to the OPC Alarms & Events Specification for a detailed description of the parametr list and return value.)***

HRESULT DisableConditionBySource (HANDLE hConnect,
DWORD dwNumSources,
LPWSTR *pszSources);

Places all conditions for the specified event sources into the disabled state. Therefore, the server will no longer generate condition-related events for these conditions.

The effect of this method is global within the scope of the event server. Therefore, if the server is supporting multiple clients, the conditions are disabled for all clients, and they will stop receiving the associated condition-related events. ***(Refer to the OPC Alarms & Events Specification for a detailed description of the parametr list and return value.)***

```
HRESULT GetFilter  (HANDLE      hConnect,
                    HANDLE      hSubscription,
                    DWORD       *pdwEventType,
                    DWORD       *pdwNumCategories,
                    DWORD       **ppdwEventCategories,
                    DWORD       *pdwLowSeverity,
                    DWORD       *pdwHighSeverity,
                    DWORD       *pdwNumAreas,
                    LPWSTR      **ppszAreaList,
                    DWORD       *pdwNumSources,
                    LPWSTR      **ppszSourceList);
```

Returns the filter currently in use for event subscriptions. ***(Refer to the OPC Alarms & Events Specification for a detailed description of the parametr list and return value.)***

```
HRESULT SetFilter (HANDLE      hConnect,
                   HANDLE      hSubscription,
                   DWORD       dwEventType,
                   DWORD       dwNumCategories,
                   DWORD       *pdwEventCategories,
                   DWORD       dwLowSeverity,
                   DWORD       dwHighSeverity,
                   DWORD       dwNumAreas,
                   LPWSTR      *pszAreaList,
                   DWORD       dwNumSources,
                   LPWSTR      *pszSourceList);
```

Sets the filtering criteria to be used for the event subscription.

Events may be selected using the following criteria:

- Type of event, i.e. simple, condition, or tracking.
- Event categories
- Lowest severity, i.e. all events with a severity greater than or equal to the specified severity.
- Highest severity, i.e. all events with a severity less than or equal to the specified severity.
- Process areas
- Event Sources

A list of values for a single criterion are logically ORed together (e.g. if two event categories are specified, event notifications for both categories will be received). If multiple criteria are specified, they will be logically ANDed together, i.e. only those events satisfying all criteria will be selected. An example is specifying both lowest severity and highest severity will result in the selection of events with severities lying between the two values.

An OPCEventSubscription object has only one filter.

(Refer to the OPC Alarms & Events Specification for a detailed description of the parametr list and return value.)

ClientTst MFC Example

The clientTst application was designed to demonstrate how an OPC Client application may be assembled using the WTclient dll. clientTst was built using MSDEV Version 6.0. All project and resource files are contained in the distribution file ClientTst.zip.

This simple application uses the standard MFC framework, (Application/Document/View), architecture generated by the Microsoft development platform for a multiple-document interface. In this example, only one Server connection is allowed at a time. Menu options allow you to select a server using the local Windows Registry. Adding an OPC Group will generate a new Document which will enable additional menu items for creating and viewing tags from the server.

ClientTst2 MFC example

The ClientTst2 MFC application was designed to demonstrate a different technique for browsing the server namespace. It was also designed using MSDEV 6.0 and the MFC framework. ClientTst2 displays event messages from the WTClient.DLL as well s the current value for each subscribed tag.

AEClient MFC example

The AEClient MFC application was designed to demonstrate a simple Alarms & Events logger. It establishes a connection to a specified server and displays alarm and event messages as they occur.